AEROLÍNEA FRBA

ESTRATEGIA

Grupo: EL PUNTERO

Número de grupo: 12

Integrantes:

- *EDELSTEIN, IARA - 149.321-8*
- *GAROFALO, LUCIANO - 149.251-2*
- *GIRARD, MATÍAS - 149.249-4*
- *JARABROVISKI, MELANIE - 149.254-8*

Profesor: MOSCUZZA, MARCELO

Índice

Estrategia

Para la realización del trabajo práctico comenzamos plasmando toda la información detallada del enunciado en un Diagrama Entidad Relación, el cual nos permitió establecer los criterios principales a tener en cuenta.
Una vez determinados los identificadores para cada una de las tablas a crear, convenimos en la utilización de claves subrogadas en lugar de claves compuestas o naturales, por considerar más fácil la uniformidad de criterios con respecto a las diferentes entidades.

Una vez terminado el DER y teniendo más claro el modelo, procedimos a la realización de la migración. Para ello nos apoyamos tanto en la consigna, como en la tabla maestra brindada. Debido a que, basándonos en los campos de la tabla maestra y sus tipos, debimos de realizar algunas consideraciones del modelo, estas serán explicadas en la parte correspondiente de este mismo archivo.

Finalizada la migración, y comprobado que no se hayan perdido datos en el transcurso de ella, comenzamos a realizar la aplicación en C#.
Sabiendo que la aplicación funcionará como puente entre el usuario y la base de datos, fue necesario primero establecer la conexión entre base de datos y aplicación.
Para esto, realizamos cuatro clases distintas. Dichas clases son: '**DBManager**' (encargada de manejar la conexión con la base de datos, que implementa el patrón Singleton para que exista una única conexión por sistema), '**DBQueries**' (encargada de manejar los nombres de todas las consultas existentes en la base de datos), '**StoreProcedure**' (encargada de impactar a la base de datos, es la que ejecuta los métodos de ExecuteNonQuery, ExecuteScalar y ExecuteReader) y '**SPParameter**' (encargada de representar a los parámetros que espera una determinada consulta).

Una vez que pudimos efectivamente comunicarnos con la base de datos, comenzamos a desarrollar las distintas funcionalidades. La primer fue el Login, que creemos que es uno de los puntos más importantes para poder comprender y desarrollar la plataforma, además de ser el primero que cronológicamente debe ver un usuario al entrar a la aplicación.

Y además esto nos permitió empezar a realizar pruebas y a familiarizarnos con elementos como DataGridViews, Queries, pasaje de parámetros, etc. testeando las conexiones y ayudándonos a acostumbrarnos a la forma de trabajo que necesitaríamos para la realización del TP.

Por último, la forma sistemática de trabajo que adoptamos fue la de dividirnos los diferentes partes de la aplicación, e ir trabajando conjuntamente las pantallas, el código de la aplicación y al mismo tiempo los Stored Procedures, Funciones o Triggers que íbamos necesitando a medida que avanzabamos.

Estructura general del proyecto

La solución cuenta con los siguientes proyectos:

- **AerolineaFrba:** es el proyecto que almacena todos los formularios necesarios para poder utilizar todos los casos de usos planteados en la consigna, como así también al archivo de configuración del sistema ('**app.config**').

- **Configuración:** es el proyecto encargado de manejar las distintas configuraciones que se encuentran en el archivo de configuración del proyecto y que necesita el sistema para funcionar. Por ejemplo, la connection string de la base de datos y la fecha del sistema.

- **Filtros:** es el proyecto que representa los datos introducidos en los filtros de búsqueda para las consultas, tanto para las búsquedas exactas (comparan por '='), como las inexactas (comparan por '**LIKE**'). Estos filtros fueron utilizados en las pantallas de ABM's como así también para la realización del listado estadístico.

- **Herramientas:** se encarga de realizar validaciones de tipos a partir de datos introducidos por el usuario, como así también encriptar la contraseña de un nuevo usuario. También incluimos una validación de formato de matrículas de aeronave.

- **Persistencia:** es el encargado de representar a cada entidad particular que identificamos en la consigna, como así también las clases encargadas de manejar a dichos objetos en la base de datos (tanto para obtenerlas, como para impactar algún cambio).

- **Sesión:** es el proyecto encargado de manejar información del usuario logueado que está ejecutando el sistema. Cuenta con la información del usuario logueado y el rol con el que eligió loguearse. Este proyecto fue muy útil ya que fue utilizado al momento de registrar el usuario (administrador) que realiza las compras y las devoluciones.

Consideraciones generales

A medida que desarrollamos la solución, ocurría que nos encontrábamos con situaciones ambiguas, de las cuales podían surgir varias interpretaciones, es por esto que realizamos el siguiente listado de consideraciones.

- A la hora de realizar los filtros de búsqueda en todas las funcionalidades que los necesitan, decidimos implementar dos tipos de búsquedas. Por un lado, una búsqueda exacta (filtrar por '**=**'), y por otro, una búsqueda inexacta (filtrar por '**LIKE**'). La primera de estas permite filtrar resultados pasándole cualquier tipo de dato (texto libre, selección acotada, selección de fechas), mientras que la última permite únicamente filtrar por tipos de datos de texto, es decir, que no permite realizar una búsqueda inexacta con valores numéricos o fechas. Para poder realizar esto, decidimos inhabilitar los controles que no cumplen con dicha condición y volver a habilitarlos en el caso que el usuario decida realizar una búsqueda exacta.
- Se pedía en la consigna que todas las bajas realizadas (en los distintos ABM's) en el sistema fueran lógicas. Para esto, fue necesario agregar una columna '**Habilitado**' a las distintas tablas donde se debían realizar eliminaciones. Cuando efectivamente se realizaba esta acción sobre alguna de dichas tablas, en vez de ejecutar una consulta '**DELETE**', efectuábamos una consulta '**UPDATE**' que lo único que realizaba era setearle el valor "*False*" al campo '**Habilitado**' del registro a eliminar.
- Consideramos que los viajes que existían en la tabla maestra ya estaban registrados como que habían llegado a destino ya que tenían una fecha de llegada y no podríamos perder ese dato en la migración.
- Asimismo, debido al ítem anterior, y para evitar inconsistencias, decidimos que aunque esos viajes ya hayan llegado, no le generaron millas a los clientes que participaron de esos viajes.

A continuación, presentamos una serie de decisiones concretas que hemos tomado a la hora de desarrollar la solución, para poder hacer frente a estas distintas situaciones problemáticas, las mismas están divididas en los distintos requerimientos:

1. **ABM de Rol**

- En esta funcionalidad, solo realizamos la búsqueda por Nombre de Rol ya que es el único campo que no es autogenerado que tiene la tabla de Rol de nuestra solución.
- No es posible la inserción de un rol sin funcionalidades, ni tampoco la modificación de un rol existente quitándole todas las funcionalidades que tenía.
- Se utilizó un trigger (explicado más abajo) para cumplir con la funcionalidad de quitar los roles inhabilitados a aquellos usuarios que lo poseen.

2. **Login y seguridad**

- Basándonos en la consigna, creamos 3 roles distintos con sus determinadas funcionalidades, además agregamos un set de usuarios con perfil administrador para probar esta funcionalidad que son los siguientes:
  Username: "admin1" Contraseña: "w23e" Rol: Administrador
  Username: "admin2" Contraseña: "w23e" Rol: Administrador

Username: “admin3” Contraseña: “w23e” Rol: Administrador
Username: “admin4” Contraseña: “w23e” Rol: Administrador
Y un usuario administrador general que tiene todas las funcionalidades del sistema:
Username: “admin” Contraseña: “w23e” Rol: AdministradorGeneral

- Para todos estos usuarios creados en el script de creación inicial, se hardcodeo la contraseña señalada ya encriptada, ya que el sistema valida las contraseñas utilizando el algoritmo SHA256 de encriptación. Para esto contamos con una clase ‘**SHA256Encriptador**’ que a partir de una cadena, retorna su valor encriptado.
- Agregamos además la posibilidad de que un usuario modifique su contraseña, esto es posible una vez que algún usuario administrador ingresa al sistema.
- Para el manejo del usuario logueado, en vez de instanciar un objeto Usuario en el momento que se loguea el usuario e ir pasando dicho objeto entre las distintas pantallas que lo necesiten, decidimos crear una clase global ‘**AdministradorSesion**’, encargada de contener la información del usuario logueado, el rol con el que se logueo y el momento de login.
- Todos los usuarios empiezan teniendo una cantidad de intentos posibles de error de contraseña igual a 3. Esta se va descontando en la medida que el usuario realice un logueo fallido. Si cumple con 3 equivocaciones, el usuario quedará inhabilitado y no es posible volver a habilitarlo.

3. Registro de Usuario

- Para los usuarios nuevos, es decir, los que se van a ir generando mediante nuestro sistema desarrollado, también utilizamos la clase ‘**SHA256Encriptador**’ que a partir de una cadena, retorna su valor encriptado. Luego, este nuevo valor calculado se le pasa por parámetro a la consulta que inserta los nuevos usuarios.
- Como en la consigna no estaba especificado, decidimos que para los nuevos usuarios ingresados, no se puede seleccionar un rol para los mismos, sino que por default el sistema los ingresa con rol Administrador.

4. AMB de Ciudad

- Si bien se había mencionado por el grupo que esta funcionalidad no era necesaria, decidimos crearla de todos modos con algunas modificaciones.
- La búsqueda de este ABM únicamente se puede realizar por el nombre de la ciudad, ya que es el único campo no autogenerado que tiene nuestra tabla Ciudad de nuestra solución.
- En este ABM, tomamos la decisión de que en caso de querer realizar una baja de una ciudad, esta no será posible si tiene viajes designados. Y en caso de que sea posible, la misma será en forma física.

5. ABM de Ruta Aérea

- Consideramos que una ruta solamente puede existir si tiene algún (uno o más) servicios que brinda. Por lo que no es posible crear una ruta sin servicios o modificar una existente sacándole todos los servicios que poseía.
- Para este ABM se utilizó una búsqueda completa, es decir, por todos los campos que tiene nuestra tabla de Ruta no autogenerados.

- Para inhabilitar una ruta se hace desde la pantalla de búsqueda (botón de Inhabilitar). Para volver a habilitarla, se debe hacer desde la parte de modificar la ruta (con el check).
- Volver a habilitar una ruta previamente inhabilitada no “deshace” las cancelaciones de pasajes y encomiendas que causó.
- En la modificación, no se valida la existencia de códigos de ruta duplicados ya que en las rutas provenientes de la tabla maestra esto es admitido. En el alta, se verifica este requerimiento.

6. ABM de Aeronave

- En este ABM realizamos la búsqueda por varios campos como matrícula, modelo, aeronave y fecha alta.
- Para buscar con el filtro de la fecha de alta de una aeronave es necesario seleccionar esa opción ya que sino siempre busca por la fecha del día.
- Siempre antes de listar las aeronaves se verifica mediante un procedure si hay aeronaves que volvieron a estar habilitadas ya que terminó su período de fuera de servicio.
- En la modificación, se pueden modificar todos los campos de una aeronave siempre y cuando no tenga viajes asignados, menos la fecha alta ya que consideramos que no es un campo que tendría sentido que varíe, a menos que esta fecha no haya llegado.
- A la hora de dar de baja una aeronave y elegir la opción de reemplazar la aeronave por otra y tiene que dar de alta una nueva para hacer esto, validamos que esta última tenga que tener como mínimo la misma cantidad de butacas de cada tipo que la aeronave a reemplazar, como así también con la cantidad de KGs.
- Para la cantidad de butacas de aeronaves ya existentes en la base de datos maestra, consideramos la cantidad de pasajes que había, en base a su número de butaca, tipo y aeronave a la que pertenecía. En cambio, la cantidad de butacas de aeronaves nuevas se ingresa manualmente e impacta directamente en la tabla de butacas. Entonces, a la hora de consultar la cantidad de butacas de una aeronave, se consulta directamente esta última tabla.
- Además en la modificación de la aeronave, si ésta es posible, es decir que no tiene viajes asignados, existe un ABM de butacas para la misma. Las butacas se pueden habilitar e inhabilitar, modificar su servicio y agregar nuevas.
- Se asume que al reemplazar una aeronave por otra, la aeronave reemplazante va a tener la cantidad de butacas necesarias para reemplazar a la otra y lo mismo para los kgs. Ya que en la consigna la única validación que había que tener en cuenta era que sea de la misma flota. Cuando hay que dar de alta una nueva, estas validaciones de cantidades se tienen que cuenta, pero en el reemplazo automático no.

7. Generación de Viaje

- A la hora de generar un viaje comenzamos estableciendo la aeronave que realizará el mismo, con lo cual se establece el servicio ya predeterminado para esta.
- Cuando se elige la ciudad origen sólo se mostrarán aquellas que pertenezcan a una ruta que contenga el servicio de la aeronave seleccionada. Como así también al momento de seleccionar el destino solo será posible optar por aquellas ciudades que tengan en su ruta a la ciudad origen establecida.

- En cuanto a la fecha de llegada, determinamos que esta se establecerá como la de llegada estimada que luego podrá modificarse con el registro de llegada a destino.

8. Registro de llegada a Destino

- Para registrar la llegada de un viaje el usuario deberá ingresar la ciudad origen, ciudad destino, y fecha de salida del mismo para poder así estar habilitado para modificar la fecha de llegada.
- Una vez validado lo mencionado anteriormente se procederá a mostrar la información de la aeronave que realiza el viaje y la opción de registrar la fecha, la cual deberá ser posterior a la fecha de salida y cumplir con que sea un tiempo de vuelo menor a 24hs.
- En esta funcionalidad también se suman las millas de pasajero frecuente de todos los clientes que viajaron o enviaron las encomiendas.
- Un viaje que estaba planeado para una aeronave, una fecha y una ciudad de origen determinados puede llegar a cualquier ciudad de destino inclusive a la misma de la que partió (esto puede ocurrir por algún imprevisto). Esta consideración se efectuó para poder validar si el avión debía arribar a esa ciudad de destino o no. Por lo tanto, el avión puede recorrer una ruta inexistente en el sistema. Si bien se puede registrar el arribo a otro destino, este cambio no impacta en la base de datos y todas las funcionalidades que de esta derivaban (por ejemplo, el registro de las millas), se ejecutan en forma normal.

9. Compra de pasaje/encomienda

- Es importante aclarar que para aquellos documentos repetidos de clientes ya existentes, el llenado automático de datos no se realiza.
- Los campos se llenan automáticamente luego de ingresar el tipo y número de documento y haciendo click en otro text box o seleccionando la butaca, o bien tocando Enter.
- Para acotar los resultados debido a la gran cantidad de viajes existentes en la base de datos, decidimos que la búsqueda de vuelos para comprar pasajes o encomiendas, sólo pueda realizarse buscando exactamente por origen, destino y fecha deseada.
- Las opciones de pago dependen del rol del usuario logueado. Si es cliente, sólo puede pagar con tarjeta, en cambio un administrador puede hacerlo tanto con tarjeta como en efectivo.
- El formulario de carga de datos de una tarjeta de crédito posee información (como por ejemplo, vencimiento o código de seguridad) que está para hacer el formulario más real. En realidad, no los procesamos ni guardamos porque entendemos que en la realidad, esos datos se compararían contra una base de datos de algún banco o empresa de tarjetas.
- El PNR es autogenerado y se le informe al cliente una vez confirmada la operación de compra.

10. Devolución/Cancelación de pasaje y/o encomienda

- Para proceder a la devolución de las compras o encomiendas de un cliente en particular el usuario deberá ingresar el dni del mismo. Una vez validada su existencia se mostrarán todas las posibles devoluciones a realizar.

- En el caso que se ingrese un número de documento que esté por duplicado en la base de datos, se le exigirá al usuario que ingrese la fecha de nacimiento del mismo para poder así identificarlo unívocamente.
- El usuario tendrá la posibilidad de devolver todo (todos los pasajes y encomiendas comprados), lo cual generará que todas sus compras sean devueltas, exigiendo que se ingrese un motivo común a todas las devoluciones.
- No se podrán realizar devoluciones de viajes que tengan una fecha de salida menor o igual a la fecha del sistema. Para el caso de los viajes que se migraron de la tabla maestra, como los mismos ya cuentan con fecha de llegada o sea que ya fueron registrados, más allá que la fecha del sistema sea anterior a la llegada, estos no podrán ser devueltos.

11. Consulta de millas de pasajero frecuente

- Para ingresar a esta funcionalidad, se le pide al Cliente/Administrador que ingrese el tipo y número de documento de quién quiere consultar sus millas. En caso que se encuentre en la base de datos dos números de documento repetidos, se le solicitará al cliente su fecha de nacimiento para su identificación.
- Como esta funcionalidad pide un registro de las millas del cliente (todos los movimientos para conseguirlas), se muestra por un lado, todas las compras del cliente (tanto pasajes como encomiendas) como la cantidad de millas que suman por esa operación. Por otro lado se muestran los canjes realizados por el cliente con todo su detalle.
- Cuando lista todos los registros con los movimientos del cliente, se van sumando las millas en caso que sea compra y restando en caso que sea canje y se muestra la totalidad de las millas con las que el cliente cuenta.
- Cada vez que se inicia la aplicación, se ejecuta un procedure que resta las millas vencidas de los clientes y a la vez, elimina sus registros de la base de datos.

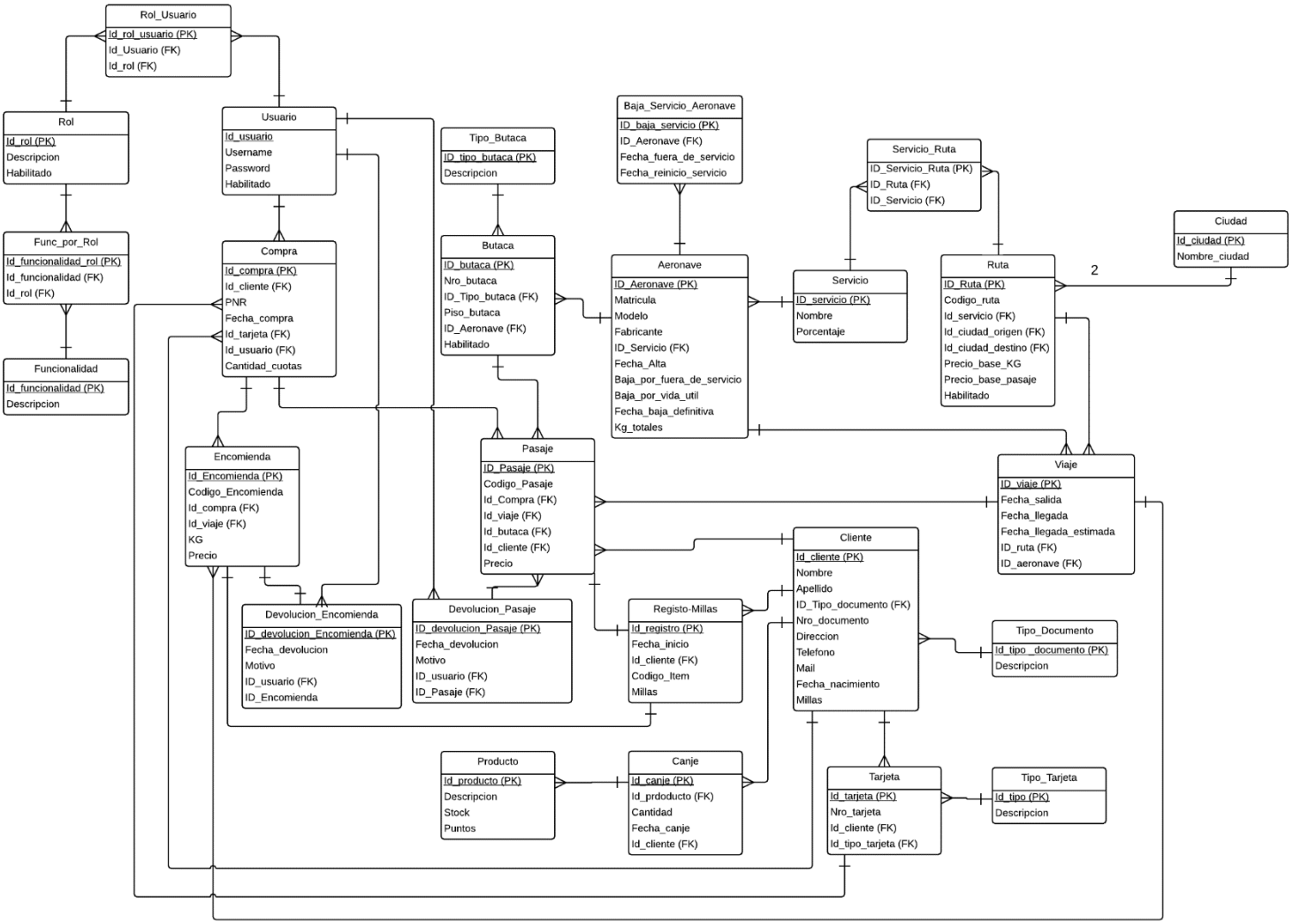
12. Canje de millas

- Para ingresar a esta funcionalidad, se le pide al Cliente/Administrador que ingrese el tipo y número de documento de quién quiere realizar el canje. En caso que se encuentre en la base de datos dos números de documento repetidos, se le solicitará al cliente su fecha de nacimiento para su identificación.
- Agregamos a la base de datos algunos productos con respectivos valores inventados de millas y cantidades de stock.
- Cuando un cliente ingresa, solo se le muestran los productos para los cuales le alcanzan sus millas para canjear. Por otro lado, si quiere visualizar todos los productos disponibles, hay una opción para ver el listado completo.

13. Listado Estadístico

- El Administrador podrá elegir el año de consulta, el semestre y la estadística que quiere visualizar.
- Tomamos como inicio de cada semestre al día 1 del mes y como fin al día que corresponda según el mes (30 o 31).
- Además de mostrar las descripciones del TOP 5 determinado para cada estadística se muestra el valor que justifica dicho posicionamiento.

DER


Rol_Usuario
Id_rol_usuario (PK)
Id_Usuario (FK)
Id_rol (FK)
Rol
Id_rol (PK)
Descripcion
Habilitado
Usuario
Id_usuario
Username
Password
Habilitado
Tipo_Butaca
ID_tipo_butaca (PK)
Descripcion
Baja_Servicio_Aeronave
ID_baja_servicio (PK)
ID_Aeronave (FK)
Fecha_fuera_de_servicio
Fecha_reinicio_servicio
Servicio_Ruta
ID_Servicio_Ruta (PK)
ID_Ruta (FK)
ID_Servicio (FK)
Ciudad
Id_ciudad (PK)
Nombre_ciudad
Func_por_Rol
Id_funcionalidad_rol (PK)
Id_funcionalidad (FK)
Id_rol (FK)
Compra
Id_compra (PK)
Id_cliente (FK)
PNR
Fecha_compra
Id_tarjeta (FK)
Id_usuario (FK)
Cantidad_cuotas
Butaca
ID_butaca (PK)
Nro_butaca
ID_Tipo_butaca (FK)
Piso_butaca
ID_Aeronave (FK)
Habilitado
Aeronave
ID_Aeronave (PK)
Matricula
Modelo
Fabricante
ID_Servicio (FK)
Fecha_Alta
Baja_por_fuera_de_servicio
Baja_por_vida_util
Fecha_baja_definitiva
Kg_totales
Servicio
ID_servicio (PK)
Nombre
Porcentaje
Ruta
ID_Ruta (PK)
Codigo_ruta
Id_servicio (FK)
Id_ciudad_origen (FK)
Id_ciudad_destino (FK)
Precio_base_KG
Precio_base_pasaje
Habilitado
2
Funcionalidad
Id_funcionalidad (PK)
Descripcion
Encomienda
Id_Encomienda (PK)
Codigo_Encomienda
Id_compra (FK)
Id_viaje (FK)
KG
Precio
Pasaje
ID_Pasaje (PK)
Codigo_Pasaje
Id_Compra (FK)
Id_viaje (FK)
Id_butaca (FK)
Id_cliente (FK)
Precio
Viaje
ID_viaje (PK)
Fecha_salida
Fecha_llegada
Fecha_llegada_estimada
ID_ruta (FK)
ID_aeronave (FK)
Cliente
Id_cliente (PK)
Nombre
Apellido
ID_Tipo_documento (FK)
Nro_documento
Direccion
Telefono
Mail
Fecha_nacimiento
Millas
Devolucion_Encomienda
ID_devolucion_Encomienda (PK)
Fecha_devolucion
Motivo
ID_usuario (FK)
ID_Encomienda
Devolucion_Pasaje
ID_devolucion_Pasaje (PK)
Fecha_devolucion
Motivo
ID_usuario (FK)
ID_Pasaje (FK)
Registo-Millas
Id_registro (PK)
Fecha_inicio
Id_cliente (FK)
Codigo_Item
Millas
Tipo_Documento
Id_tipo _documento (PK)
Descripcion
Producto
Id_producto (PK)
Descripcion
Stock
Puntos
Canje
Id_canje (PK)
Id_prdoducto (FK)
Cantidad
Fecha_canje
Id_cliente (FK)
Tarjeta
Id_tarjeta (PK)
Nro_tarjeta
Id_cliente (FK)
Id_tipo_tarjeta (FK)
Tipo_Tarjeta
Id_tipo (PK)
Descripcion

Consultas

A la hora de realizar las consultas, tras investigar sobre el motor de la base de datos y cómo conectarnos con este desde el C#, averiguamos que hay dos tipos de consultas: las que están almacenadas directamente en la base de datos y se ejecutan desde el código (stored procedures/command type) o las que están escritas directamente en el código y se ejecutan desde él mismo (command type). Aquí es donde tuvimos que tomar la primer decisión, cuál de los dos tipos elegir. Tras ver que era más cómodo y que agregado a esto, se debía entregar un archivo SQL que se encargue de preparar todo el entorno de la base de datos (tablas, consultas, disparadores), decidimos volcarnos hacia el lado de las consultas almacenadas en la base de datos.
Una vez que ya sabíamos cómo y dónde debíamos almacenarlas, tuvimos que empezar a desarrollarlas. No tomamos ninguna decisión en particular para hacer esto, sino que a medida que cada miembro del grupo encontraba que era necesario crear una nueva query, la agregaba al script inicial en su posición adecuada para que nada realizado anteriormente deje de funcionar.

Generalmente, todas las tablas cuentan con las consultas básicas: SELECT, INSERT, UPDATE y DELETE.
Realizamos tres tipos de ejecuciones de consultas desde el sistema:

- **ExecuteNonQuery:** las que realizan la inserción/modificación/baja de registros de la base de datos y retornan la cantidad de registros afectados.
- **ExecuteScalar:** las que realizan la inserción/modificación/baja de registros de la base de datos y retornan la primera columna de la primera fila del conjunto de resultados devuelto por la consulta.
- **ExecuteReader:** las que únicamente obtienen datos de la base de datos.

Triggers

Realizamos un trigger **'Tr_DeshabilitarUsuariosConRolDeshabilitado'** para realizar el borrado automático de la tabla asociativa de roles y usuarios, disparado luego de la modificación de un rol que fue deshabilitado.